AI Assistant Coding

Assignment 1.2

Name : GOLI NITHYA **HT. No :** 2303A52066

Batch: 32

Task 1: AI-Generated Logic Without Modularization (Factorial without Functions)

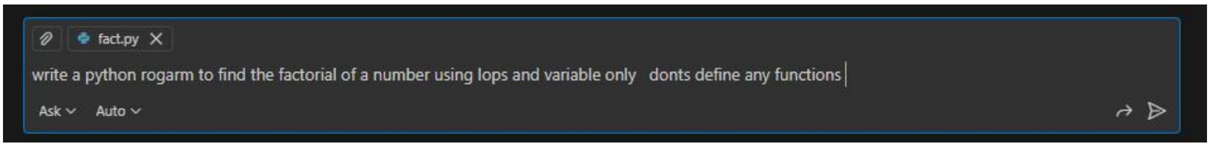
Scenario: You are building a **small command-line utility** for a startup intern onboarding task. The program is simple and must be written quickly without modular design.

Task Description: Use GitHub Copilot to generate a Python program that computes a mathematical product-based value (factorial-like logic) directly in the main execution flow, without using any user-defined functions.=

Constraint:

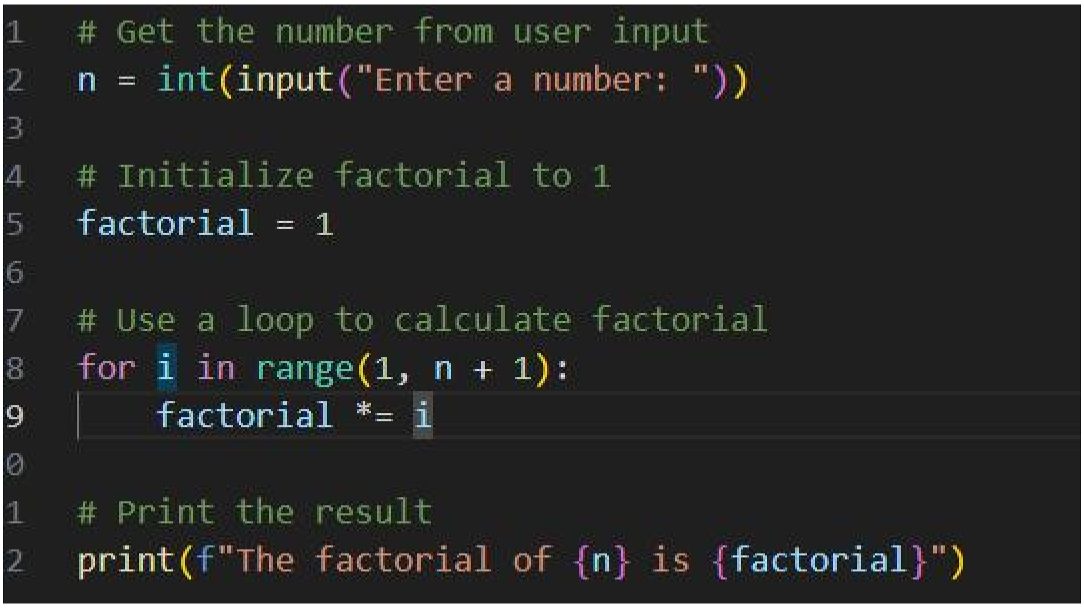
- Do not define any custom function
- Logic must be implemented using loops and variables only

Prompt:



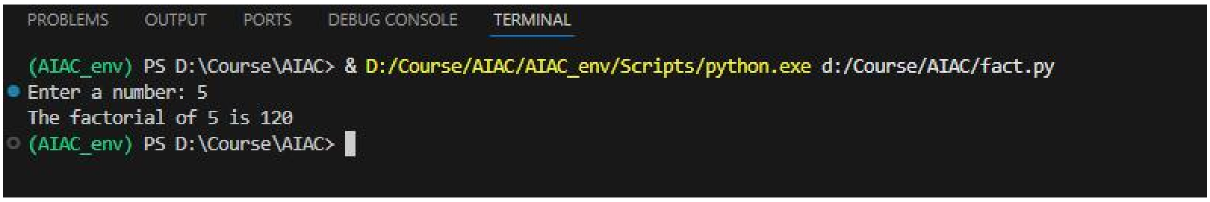
write a python rogarm to find the factorial of a number using lops and variable only   donts define any functions

Code:



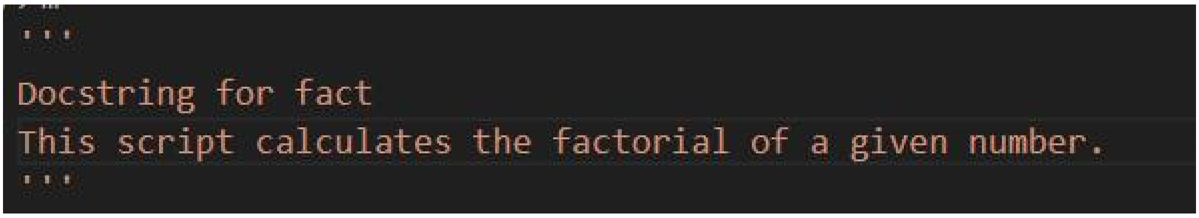
```python
# Get the number from user input
n = int(input("Enter a number: "))

# Initialize factorial to 1
factorial = 1

# Use a loop to calculate factorial
for i in range(1, n + 1):
    factorial *= i

# Print the result
print(f"The factorial of {n} is {factorial}")
```

Output:



```
(AIAC_env) PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/python.exe d:/Course/AIAC/fact.py
Enter a number: 5
The factorial of 5 is 120
(AIAC_env) PS D:\Course\AIAC> 
```

Copilot Suggestions:



```
'''
Docstring for fact
This script calculates the factorial of a given number.
'''
```

Explanation:

The code generated from compile was well structured and for beginner it was very helpful. It reduces coding time and gives code without and logical or syntax error.

Task 2: AI Code Optimization & Cleanup (Improving Efficiency)

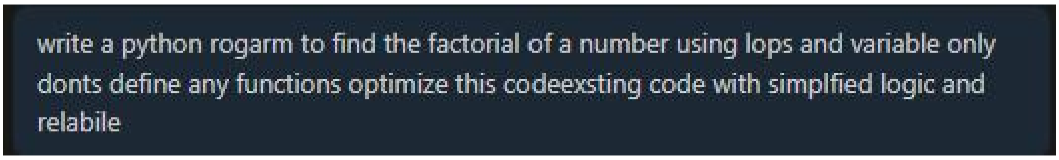
Scenario: Your team lead asks you to **review AI-generated code** before committing it to a shared repository.

Task Description : Analyze the code generated in **Task 1** and use Copilot again to:

- ➢ Reduce unnecessary variables
- ➢ Improve loop clarity
- ➢ Enhance readability and efficiency

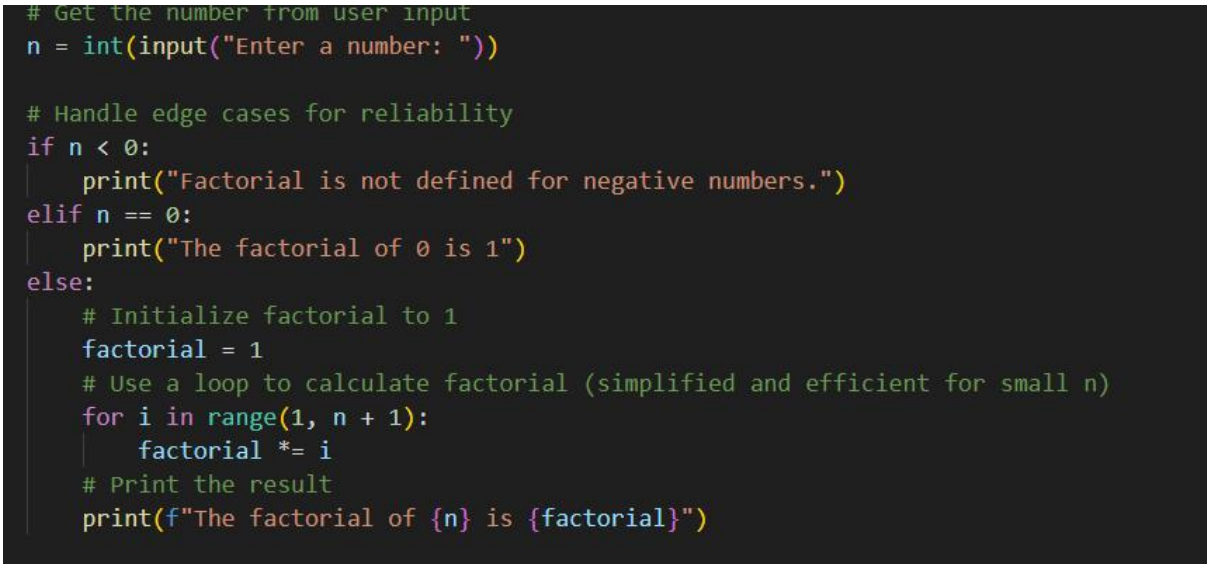
Hint: Prompt Copilot with phrases like
“optimize this code”, *“simplify logic”*, or *“make it more readable”*

Prompt:



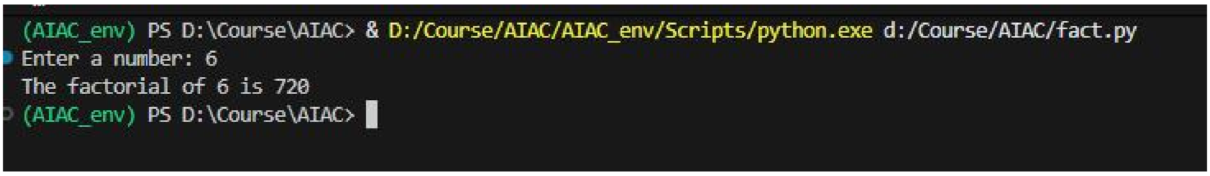
write a python rogarm to find the factorial of a number using lops and variable only donts define any functions optimize this codeexsting code with simplfied logic and relabile

Code:



```python
# Get the number from user input
n = int(input("Enter a number: "))

# Handle edge cases for reliability
if n < 0:
    print("Factorial is not defined for negative numbers.")
elif n == 0:
    print("The factorial of 0 is 1")
else:
    # Initialize factorial to 1
    factorial = 1
    # Use a loop to calculate factorial (simplified and efficient for small n)
    for i in range(1, n + 1):
        factorial *= i
    # Print the result
    print(f"The factorial of {n} is {factorial}")
```

Output:



```
(AIAC_env) PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/python.exe d:/Course/AIAC/fact.py
Enter a number: 6
The factorial of 6 is 720
(AIAC_env) PS D:\Course\AIAC> 
```

Explanation:

Compared to previous code this code was most efficient as it handles the edge case for negative and zero input

---

Task 3: Modular Design Using AI Assistance (Factorial with Functions)

Scenario: The same logic now needs to be reused in **multiple scripts**.

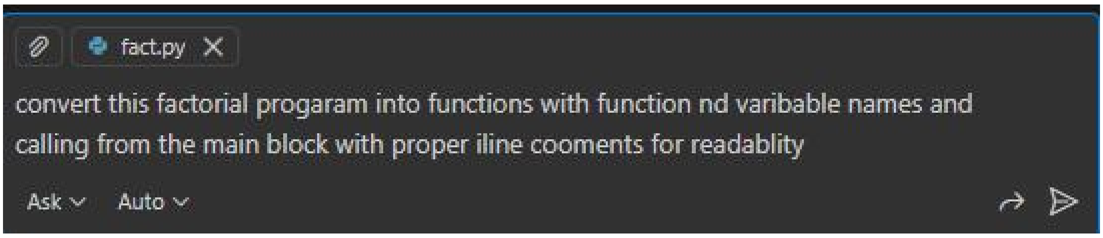
Task Description: Use GitHub Copilot to generate a **modular version** of the program by:

- Creating a **user-defined function**
- Calling the function from the main block

Constraints

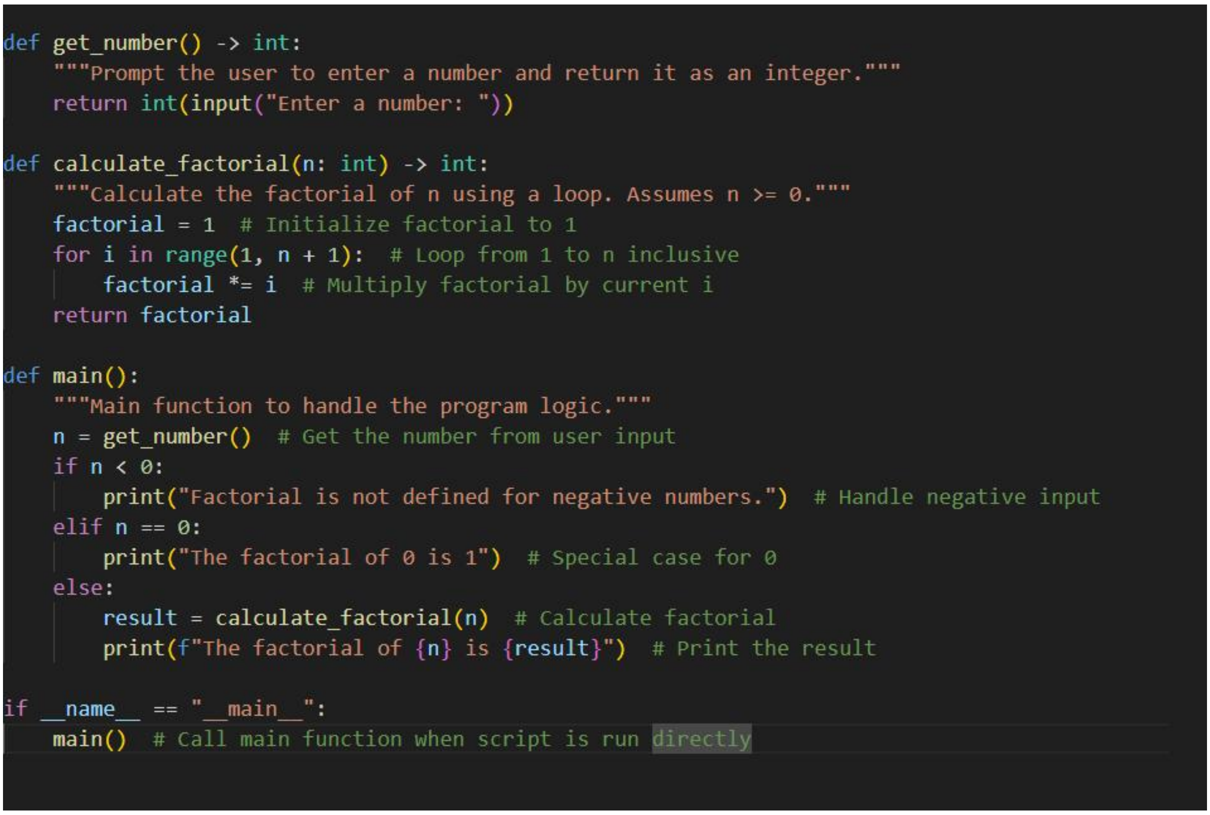
- Use meaningful function and variable names
- Include inline comments (preferably suggested by Copilot)

Prompt:



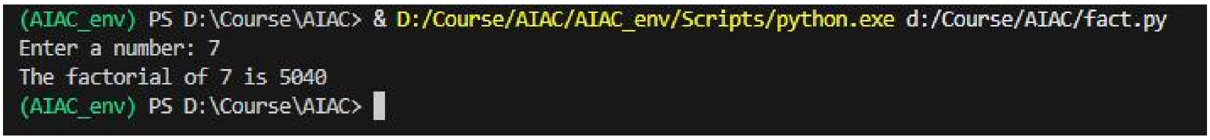
convert this factorial progaram into functions with function nd varibable names and calling from the main block with proper iline cooments for readablity

Code:



```
def get_number() -> int:
    """Prompt the user to enter a number and return it as an integer."""
    return int(input("Enter a number: "))

def calculate_factorial(n: int) -> int:
    """Calculate the factorial of n using a loop. Assumes n >= 0."""
    factorial = 1  # Initialize factorial to 1
    for i in range(1, n + 1):  # Loop from 1 to n inclusive
        factorial *= i  # Multiply factorial by current i
    return factorial

def main():
    """Main function to handle the program logic."""
    n = get_number()  # Get the number from user input
    if n < 0:
        print("Factorial is not defined for negative numbers.")  # Handle negative input
    elif n == 0:
        print("The factorial of 0 is 1")  # Special case for 0
    else:
        result = calculate_factorial(n)  # Calculate factorial
        print(f"The factorial of {n} is {result}")  # Print the result

if __name__ == "__main__":
    main()  # Call main function when script is run directly
```

Output:



```
(AIAC_env) PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/python.exe d:/Course/AIAC/fact.py
Enter a number: 7
The factorial of 7 is 5040
(AIAC_env) PS D:\Course\AIAC> 
```

Explanation:

The code is converted into different functions with main block calling and improving readability

---

Task 4: Comparative Analysis – Procedural vs Modular AI Code (With vs Without Functions)

Scenario

As part of a **code review meeting**, you are asked to justify design choices.

Task Description

Compare the **non-function** and **function-based** Copilot-generated programs on the following criteria:

- Logic clarity
- Reusability
- Debugging ease
- Suitability for large projects
- AI dependency risk

	Without Functions	**With Functions**
Logic Clarity	Logic is written in a single flow,easy to understand but becomes unreadable as lines increases.	Logic is divided into functions, making the code to read & understand easily.
Reusability	Code cannot be reused	Functions can be reused in other programs without writing the logic again.

Debugging Ease	Debugging becomes hard as all the logic at one place	Debugging becomes easy as all the logic written multiple functions
Suitability for Large Projects	Not suitable for large projects	Suitable for large projects due to proper structure
AI Dependency Risk	Higher risk for long procedural code, hard to review or modify.	Lower risk generated in functions, easy to review & Modify.

Task 5: AI-Generated Iterative vs Recursive Thinking

Scenario: Your mentor wants to test how well AI understands different computational paradigms.

Task Description: Prompt Copilot to generate:

An **iterative** version of the logic

A **recursive** version of the same logic

Constraints : Both implementations must produce identical outputs

Students must **not manually write the code first**

- **Expected Deliverables**

Two AI-generated implementations

Execution flow explanation (in your own words)

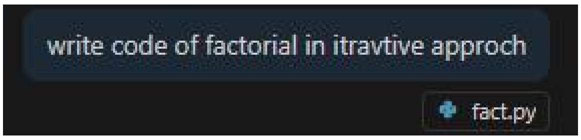
Comparison covering:

  - Readability
  - Stack usage
  - Performance implications

When recursion is *not* recommended

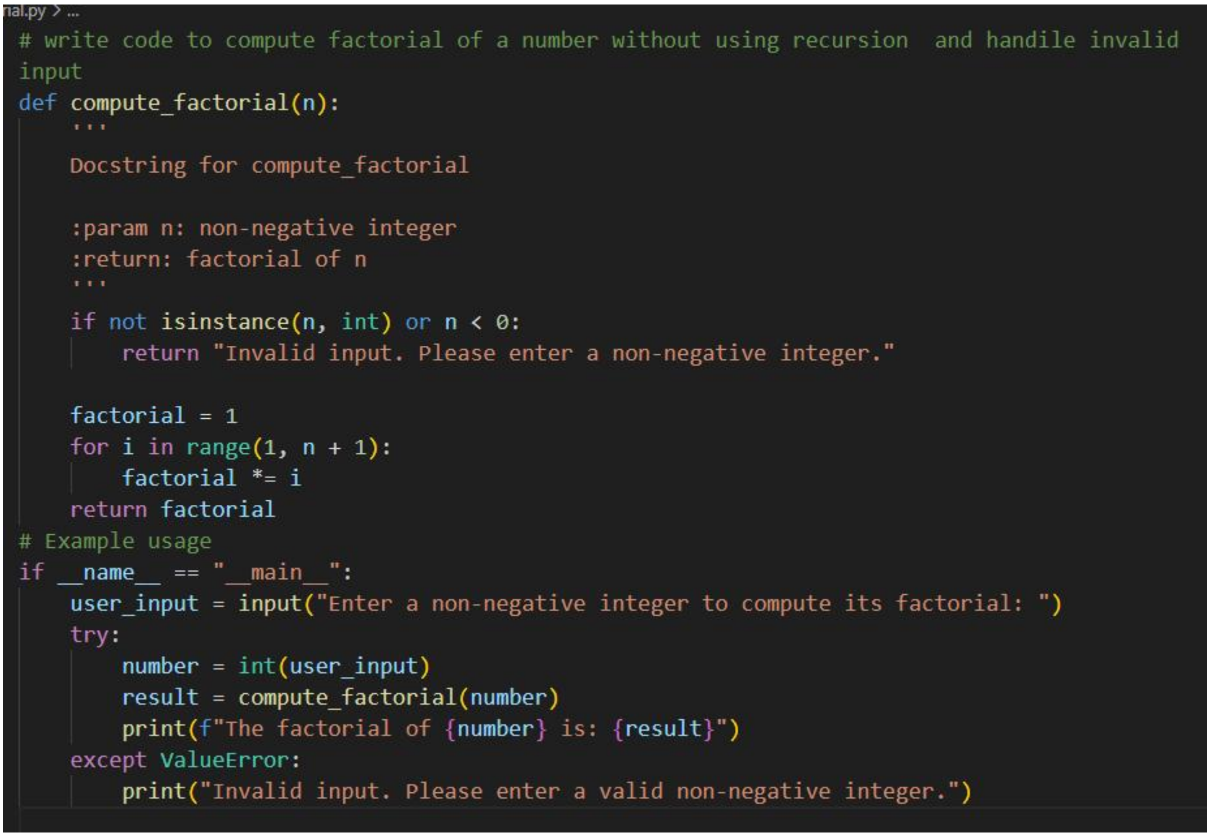
Iterative:

Prompt:



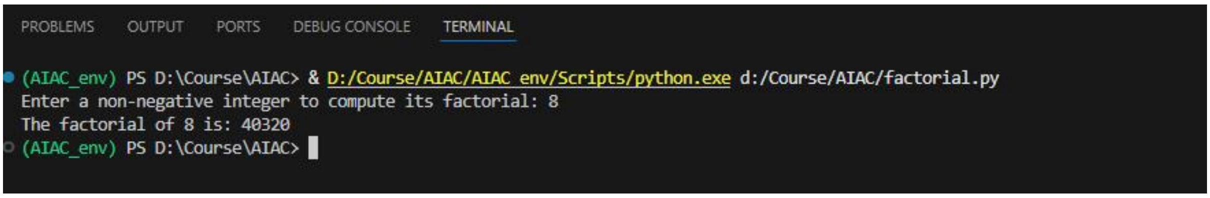
write code of factorial in itravtive approch

Code:



```python
ial.py > ...
# write code to compute factorial of a number without using recursion  and handile invalid
input
def compute_factorial(n):
    '''
    Docstring for compute_factorial

    :param n: non-negative integer
    :return: factorial of n
    '''
    if not isinstance(n, int) or n < 0:
        return "Invalid input. Please enter a non-negative integer."

    factorial = 1
    for i in range(1, n + 1):
        factorial *= i
    return factorial
# Example usage
if __name__ == "__main__":
    user_input = input("Enter a non-negative integer to compute its factorial: ")
    try:
        number = int(user_input)
        result = compute_factorial(number)
        print(f"The factorial of {number} is: {result}")
    except ValueError:
        print("Invalid input. Please enter a valid non-negative integer.")
```

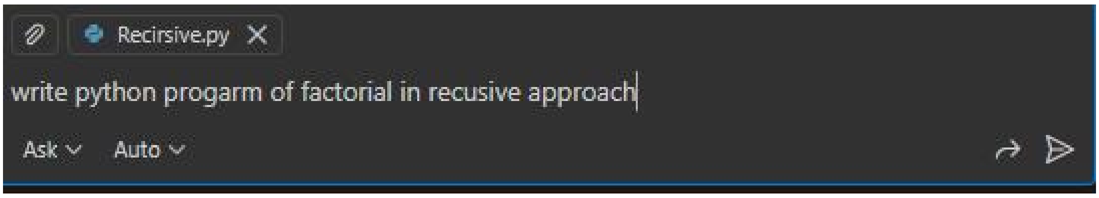
Output:



```
(AIAC_env) PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/python.exe d:/Course/AIAC/factorial.py
Enter a non-negative integer to compute its factorial: 8
The factorial of 8 is: 40320
(AIAC_env) PS D:\Course\AIAC> 
```

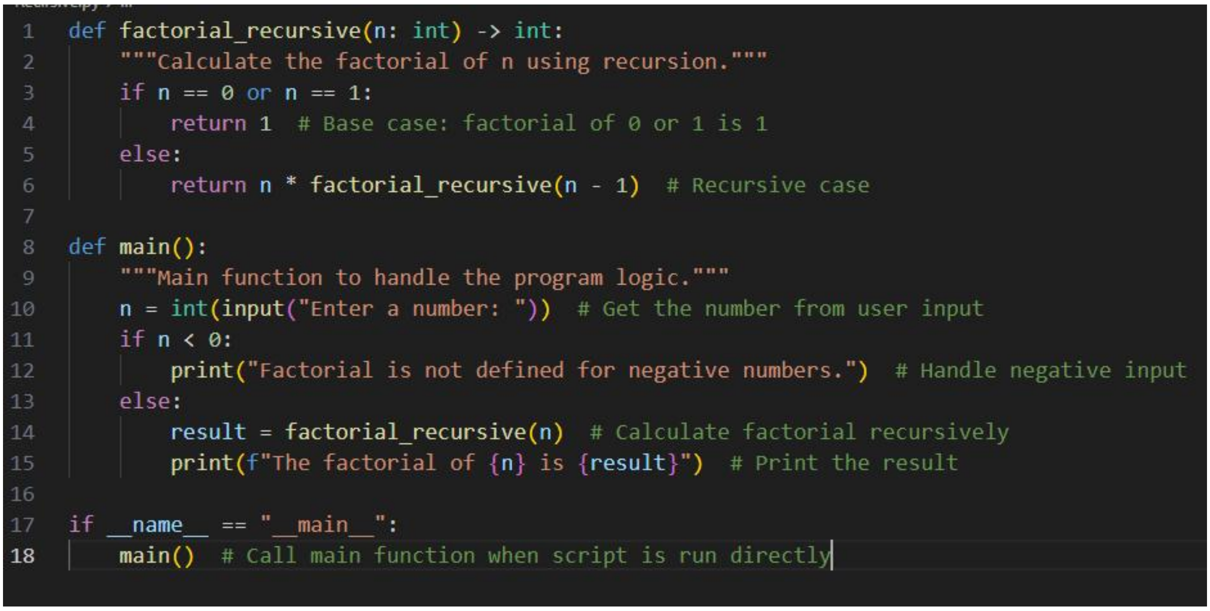
Recursive:

Prompt:



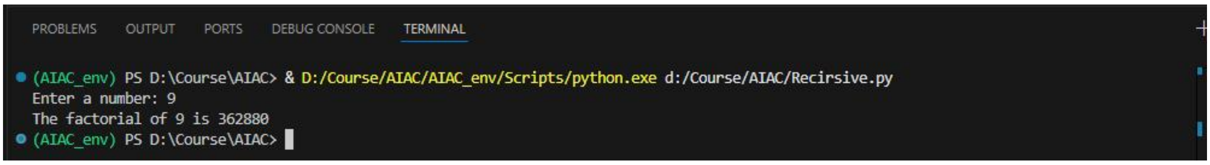
write python progarm of factorial in recusive approach

Code:



```python
def factorial_recursive(n: int) -> int:
    """Calculate the factorial of n using recursion."""
    if n == 0 or n == 1:
        return 1  # Base case: factorial of 0 or 1 is 1
    else:
        return n * factorial_recursive(n - 1)  # Recursive case

def main():
    """Main function to handle the program logic."""
    n = int(input("Enter a number: "))  # Get the number from user input
    if n < 0:
        print("Factorial is not defined for negative numbers.")  # Handle negative input
    else:
        result = factorial_recursive(n)  # Calculate factorial recursively
        print(f"The factorial of {n} is {result}")  # Print the result

if __name__ == "__main__":
    main()  # Call main function when script is run directly
```

Output:



```
(AIAC_env) PS D:\Course\AIAC> & D:/Course/AIAC/AIAC_env/Scripts/python.exe d:/Course/AIAC/Recirsive.py
Enter a number: 9
The factorial of 9 is 362880
(AIAC_env) PS D:\Course\AIAC>
```

	Iterative Approach	**Recursive Approach**
Readability	Easy to understand for beginners	harder for beginners due to function calls
Stack Usage	Uses constant memory	Uses call stack for each function call
Performance	Faster and more memory-efficient	slower due to function call overhead